React Loadable:

Code Splitting with Server Side Rendering

About me

- Former Senior Frontend Developer at Oppex
- Tech Lead at Toughbyte Ltd
- React

- github.com/**northerneyes**
- medium.com/@**northerneyes**
- twitter.com/**nordfinn**

Agenda

- Problem statement
- Code-Splitting
- Server Side Rendering
- Code-Splitting + Server Side Rendering
- Universal Data Fetching

Big applications

Loading time is big

Why is it important?

Usability

0-100ms Instant perception
100-300ms Small perceptible delay
300-1000ms Machine is working
1000+ ms Likely mental context switch
10000+ ms Task is abandoned

Performance

PageSpeed Insights

http://www.google.com/

ANALYZE

Mobile | Desktop

99 / 100 Speed

Consider Fixing:

Minify JavaScript

Show how to fix

9 Passed Rules

Show details

Download optimized image, JavaScript, and CSS resources for this page.

100 / 100 User Experience

Congratulations! No issues found.

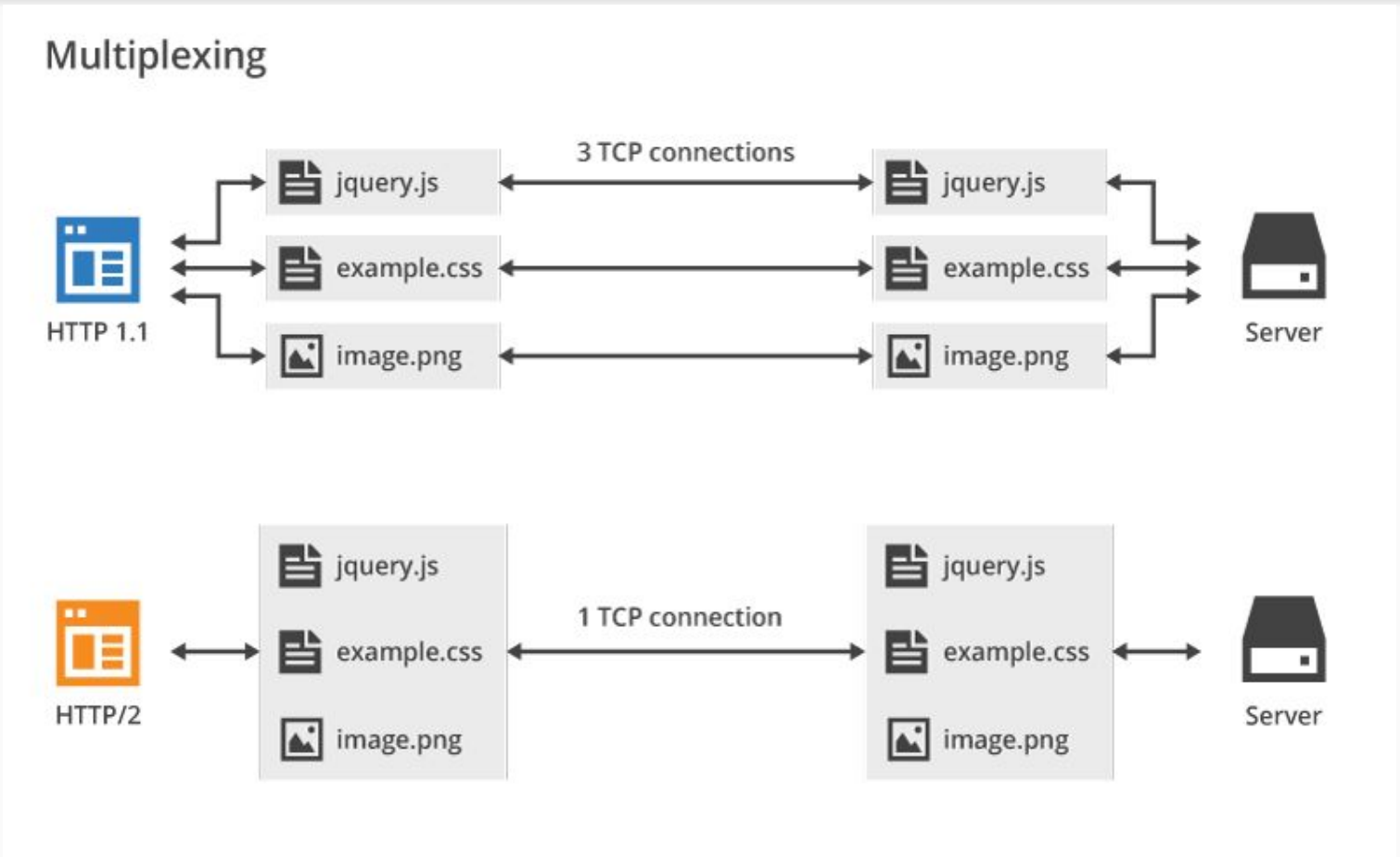
Avoid plugins

Your page does not appear to use plugins, which would prevent content from being usable on many platforms. Learn more about the importance of avoiding plugins.

Configure the viewport

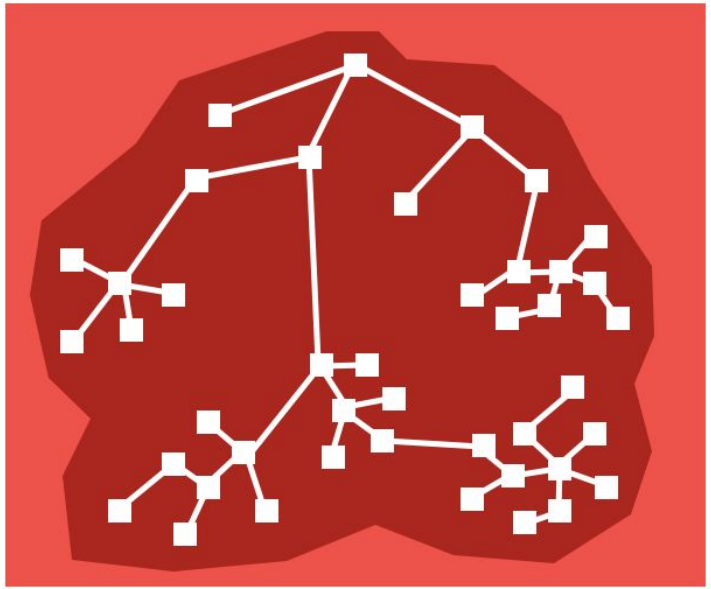
Your page specifies a viewport matching the device's size, which allows it to render properly on all devices. Learn more about configuring viewports.

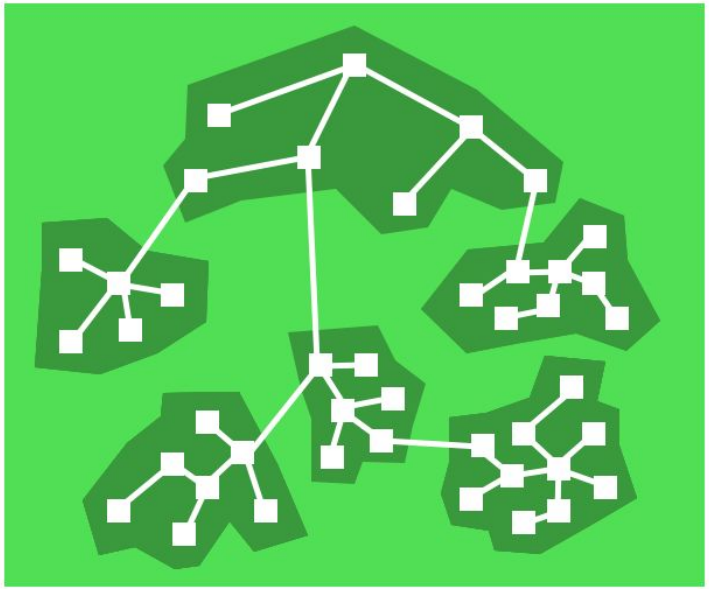
HTTP2


Multiplexing
3 TCP connections
jquery.js
example.css
image.png
jquery.js
example.css
image.png
HTTP 1.1
Server
jquery.js
example.css
image.png
1 TCP connection
jquery.js
example.css
image.png
HTTP/2
Server

Solution: Code Splitting

What is it?





Instruments:

webpack

&

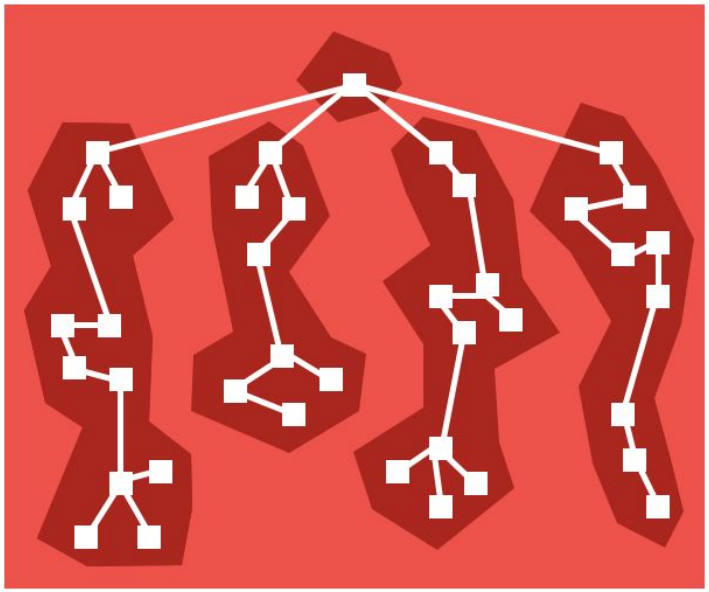
browserify

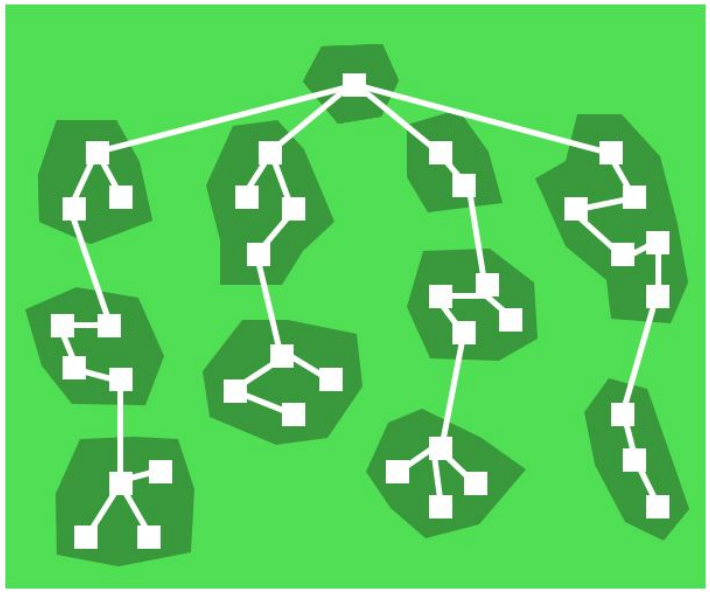
Demo

Route-based splitting vs Component-based splitting

We can do better

Component-based splitting





What is advantage?

More places to split apart our app:

- Modals
- Tabs
- Some hidden content
- And many more UI

But you can use routing as well

React Loadable

What is React Loadable?

Small library created by @thejameskyle

Loadable is Higher-Order Component (HoC)

Example

```
import AnotherComponent from './another-component';

class MyComponent extends React.Component {
  render() {
    return <AnotherComponent/>;
  }
}
```

AnotherComponent
being imported synchronously via
import

Let’s make it loaded asynchronously

```
class MyComponent extends React.Component {
  state = {
    AnotherComponent: null
  };

  componentWillMount() {
    import('./another-component').then(AnotherComponent => {
      this.setState({ AnotherComponent });
    });
  }

  render() {
    let {AnotherComponent} = this.state;
    if (!AnotherComponent) {
      return <div>Loading...</div>;
    } else {
      return <AnotherComponent/>;
    };
  }
}
```

A bunch of manual work

Loadable is simple

Loadable

```
import Loadable from 'react-loadable';

function MyLoadingComponent() {
  return <div>Loading...</div>;
}

const LoadableAnotherComponent = Loadable({
  loader: () => import('./another-component'),
  LoadingComponent: MyLoadingComponent
});

class MyComponent extends React.Component {
  render() {
    return <LoadableAnotherComponent/>;
  }
}
```

What about error handling?

Error handling is simple

```
function MyLoadingComponent({ error }) {
  if (error) {
    return <div>Error!</div>;
  } else {
    return <div>Loading...</div>;
  }
}
```

Automatic code-splitting on import()

Thanks to

webpack

Demo

Avoiding Flash Of Loading Component

Do you remember the first slide?

0-100ms Instant perception

100-300ms Small perceptible delay

300-1000ms Machine is working

1000+ ms Likely mental context switch

10000+ ms Task is abandoned

Avoiding Flash Of Loading Component

```
export default function MyLoadingComponent({ error, pastDelay }) {
  if (error) {
    return <div>Error!</div>;
  } else if (pastDelay) {
    return <div>Loading...</div>;
  } else {
    return null;
  }
}
```

Easy to change delay parameter

```
Loadable({
  loader: () => import('./another-component'),
  LoadingComponent: MyLoadingComponent,
  delay: 300
});
```

What else we can do?

What about *Preloading* ?

Preloading

```
let LoadableMyComponent = Loadable({
  loader: () => import('./another-component'),
  LoadingComponent: MyLoadingComponent,
});

class MyComponent extends React.Component {
  state = { showComponent: false };

  onClick = () => {
    this.setState({ showComponent: true });
  };

  onMouseOver = () => {
    LoadableMyComponent.preload();
  };

  render() {
    return (
      <div>
        <button onClick={this.onClick} onMouseOver={this.onMouseOver}>
          Show loadable component
        </button>
        {this.state.showComponent && <LoadableMyComponent/>}
      </div>
    )
  }
}
```

Server Side Rendering

What is Isomorphic Rendering?

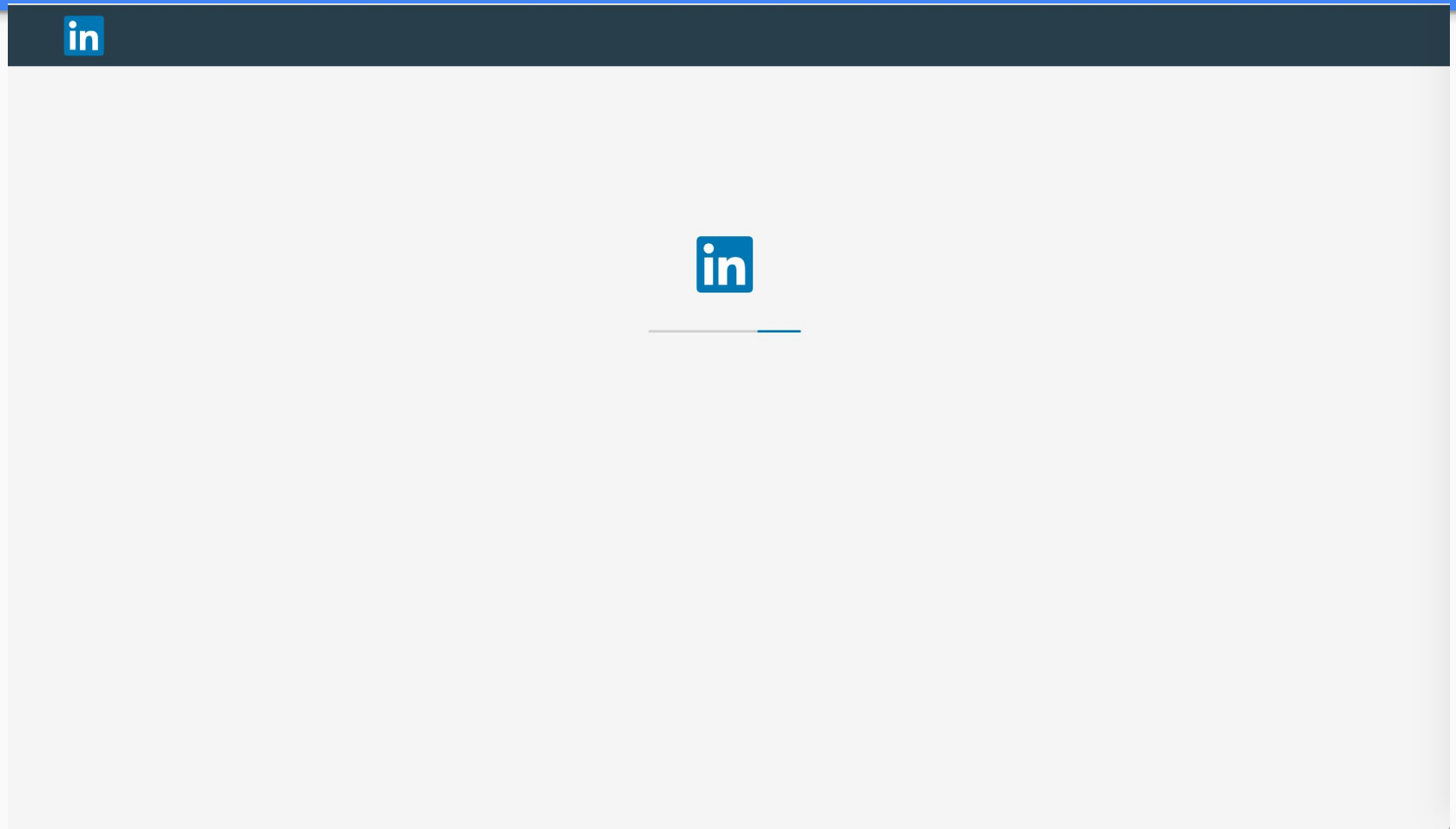
Rendering the web app on server and sending the complete HTML to the client.

The client creates the HTML in memory(Virtual Dom), checks if there are changes and re-renders the page on client.

Advantages

Better UX as user gets the complete page on first hit to the server.

Everybody like this picture :)



Better SEO as the bots can easily index the pages

Let’s make google happy

PageSpeed Insights

http://www.google.com/

ANALYZE

Mobile | Desktop

99 / 100 Speed

Consider Fixing:

Minify JavaScript

Show how to fix

9 Passed Rules

Show details

Download optimized image, JavaScript, and CSS resources for this page.

100 / 100 User Experience

Congratulations! No issues found.

Avoid plugins

Your page does not appear to use plugins, which would prevent content from being usable on many platforms. Learn more about the importance of avoiding plugins.

Configure the viewport

Your page specifies a viewport matching the device's size, which allows it to render properly on all devices. Learn more about configuring viewports.

Server Side Rendering with React is easy

React Server Rendering

```
import ReactDOMServer from 'react-dom/server';

ReactDOMServer.renderToString(
    <Provider store={store}>
      <StaticRouter location={req.url} context={{}}>
        <App />
      </StaticRouter>
    </Provider>,
  );
```

Code-splitting + Server rendering

From react-router page:

We’ve tried and failed a couple of times. What we learned:

- ❏ You need synchronous module resolution on the server so you can get those bundles in the initial render.
- ❏ You need to load all the bundles in the client that were involved in the server render before rendering so that the client render is the same as the server render. (The trickiest part, I think its possible but this is where I gave up.)
- ❏ You need asynchronous resolution for the rest of the client app’s life.

How we can solve these problems?

Let’s go back to React Loadable

Loadable

```
import Loadable from 'react-loadable';

function MyLoadingComponent() {
  return <div>Loading...</div>;
}

const LoadableAnotherComponent = Loadable({
  loader: () => import('./another-component'),
  LoadingComponent: MyLoadingComponent
});

class MyComponent extends React.Component {
  render() {
    return <LoadableAnotherComponent/>;
  }
}
```

From react-router page:

We've tried and failed a couple of times. What we learned:

- ❏ You need synchronous module resolution on the server so you can get those bundles in the initial render.
- ❏ You need to load all the bundles in the client that were involved in the server render before rendering so that the client render is the same as the server render. (The trickiest part, I think its possible but this is where I gave up.)
- ✔ You need asynchronous resolution for the rest of the client app's life.

From react-router page:

We've tried and failed a couple of times. What we learned:

- ❑ You need synchronous module resolution on the server so you can get those bundles in the initial render.
- ❑ You need to load all the bundles in the client that were involved in the server render before rendering so that the client render is the same as the server render. (The trickiest part, I think its possible but this is where I gave up.)
- ✔ You need asynchronous resolution for the rest of the client app's life.

Synchronous module resolution on the server

Synchronous loading for the server

```
import path from 'path';

const LoadableAnotherComponent = Loadable({
  loader: () => import('./another-component'),
  LoadingComponent: MyLoadingComponent,
  delay: 200,
  serverSideRequirePath: path.join(__dirname, './another-component')
});
```

From react-router page:

We’ve tried and failed a couple of times. What we learned:

- ✔ You need synchronous module resolution on the server so you can get those bundles in the initial render.
- ❑ You need to load all the bundles in the client that were involved in the server render before rendering so that the client render is the same as the server render. (The trickiest part, I think its possible but this is where I gave up.)
- ✔ You need asynchronous resolution for the rest of the client app’s life.

From react-router page:

We’ve tried and failed a couple of times. What we learned:

- ✔ You need synchronous module resolution on the server so you can get those bundles in the initial render.
- ❑ You need to load all the bundles in the client that were involved in the server render before rendering so that the client render is the same as the server render. (The trickiest part, I think its possible but this is where I gave up.)
- ✔ You need asynchronous resolution for the rest of the client app’s life.

Yes, This is the most complicated

We need 2 things

```
webpack --json > output-webpack-stats.json
```

Webpack stats about our bundle

```
let webpackStats = require('./output-webpack-stats.json');

let modules = {};
let bundles = {};

webpackStats.modules.forEach(module => {
  let parts = module.identifier.split('!');
  let filePath = parts[parts.length - 1];
  modules[filePath] = module.chunks;
});

webpackStats.chunks.forEach(chunk => {
  bundles[chunk.id] = chunk.files;
});
```

And special function flushServerSideRequires from react-loadable

```
import {flushServerSideRequires} from 'react-loadable';

let app = ReactDOMServer.renderToString(<App/>);
let requires = flushServerSideRequires();
let scripts = ['bundle-main.js'];

requires.forEach(file => {
  let matchedBundles = modules[file + '.js'];
  matchedBundles.forEach(bundle => {
    bundles[bundle].forEach(script => {
      scripts.unshift(script);
    });
  });
});
```

And the result

```
  res.send(`
    <!doctype html>
    <html>
      <head>
        <meta charset="utf-8">
        <title>react-loadable-example</title>
      </head>
      <body>
        <div id="root">${app}</div>
        ${scripts.map(script => {
          return `<script type="text/javascript" src="scripts/${script}"></script>`
        }).join('\n')}
      </body>
    </html>
  `)
})
```

Demo

Universal Data Fetching (Redux)

Fetching is simple

```
class News extends React.Component {
  constructor(props) {
    super(props);
    props.getNews();
  }

  render() {
    const { items } = this.props.news;

    return (
      ....
    );
  }
}
```

High order function

```
export default function fetch(fn) {
  return (WrappedComponent) => {
    class FetchOnLoad extends React.Component {
      constructor(props) {
        fn(this.context.store);
      }

      render() {
        return (
          <WrappedComponent {...this.props} />
        );
      }
    }

    return FetchOnLoad;
  };
}
```

But *constructor* is called both on server and client

Let’s change it

```
export default function fetch(fn) {
  return (WrappedComponent) => {
    class FetchOnLoad extends React.Component {
      componentDidMount(props) {
        fn(this.context.store);
      }

      render() {
        return (
          <WrappedComponent {...this.props} />
        );
      }
    }

    FetchOnLoad.fetch = fn;

    return FetchOnLoad;
  };
}
```

On the server

```
components.filter(component => Boolean(component && component.fetch))
      .map(component => component.fetch(store))
```

The problem to have information about these *components*

The solution is simple: let’s avoid dealing with components at all

Universal fetch

```
export default function fetch(fn) {
  const fetch = props => store => fn(store, props)

  return (WrappedComponent) => {
    class FetchOnLoad extends React.Component {

      constructor(props, context) {
        super(props);
        if (context.fetches) {
          context.fetches.push(fetch(props));
        }
      }

      componentDidMount() {
        if (!window.__INITIAL_STATE__) {
          fn(this.context.store, this.props);
        }
      }

      render() {
        return (
          <WrappedComponent {...this.props} />
        );
      }
    }
    return FetchOnLoad
    ...
```

On the server

```
function renderApp(store, req, fetches) {
  return ReactDOMServer.renderToString(
    <Provider store={store}>
      <FetchProvider fetches={fetches}>
        <App />
      </FetchProvider>
    </Provider>,
  );
}

...
// First render to collect all fetches
const fetches = [];
renderApp(store, req, fetches);

const promises = fetches.map(fetch => fetch(store));
await Promise.all(promises);
```

FetchProvider

```
export default class FetchProvider extends React.Component {
  getChildContext() {
    return {
      fetches: this.props.fetches
    };
  }

  render() {
    return this.props.children;
  }
}

FetchProvider.propTypes = {
  children: PropTypes.node.isRequired,
  fetches: PropTypes.array
};

FetchProvider.childContextTypes = {
  fetches: PropTypes.array
};
```

Demo

Conclusions

- We’ve organized Code-Splitting + Server Side Rendering for big React application
- Plus added Universal Data Fetching

Links

Main Project Link (https://github.com/northerneyes/react-stack-playground/tree/fetching)

Webpack hot server middleware (https://github.com/60frames/webpack-hot-server-middleware)

Interesting github account (https://github.com/faceyspacey)

Other Links

Usability Engineering, Jakob Nielsen, 1993 (https://www.nngroup.com/books/usability-engineering/)

Page Insights (https://developers.google.com/speed/pagespeed/insights/)

HTTP/2 (https://en.wikipedia.org/wiki/HTTP/2)

Webpack 2 documentation (https://webpack.js.org/)

React Loadable (https://github.com/thejameskyle/react-loadable)

Medium article (https://medium.com/@thejameskyle/react-loadable-2674c59de178)

Alternatives (https://github.com/ctrlplusb/react-async-component)

React router v4 (https://reacttraining.com/react-router/web/guides/code-splitting/code-splitting-server-rendering)

Webpack 2 code-splitting (https://webpack.js.org/guides/code-splitting-async/)

Thanks!

George Bukhanov

- github.com/**northerneyes**
- medium.com/@**northerneyes**
- twitter.com/**nordfinn**

React Loadable:

Code Splitting with Server Side Rendering